Model-Driven Sparse CP Decomposition for Higher-Order Tensors

Jiajia Li[1], Jee Choi[2], Ioakeim Perros[1], Jimeng Sun[1], Richard Vuduc[1]
[1] Computational Science and Engineering, Georgia Institute of Technology, Atlanta, GA, USA
[2] IBM Thomas J. Watson Research Center, Yorktown Heights, NY, USA
Email: jiajiali@gatech.edu

***Abstract*—Given an input tensor, its CANDECOMP/PARAFAC decomposition (or CPD) is a low-rank representation. CPDs are of particular interest in data analysis and mining, especially when the data tensor is sparse and of higher order (dimension). This paper focuses on the central bottleneck of a CPD algorithm, which is evaluating a sequence of matricized tensor times Khatri-Rao products (MTTKRPs). To speed up the MTTKRP sequence, we propose a novel, adaptive tensor memoization algorithm, ADATM. Besides removing redundant computations within the MTTKRP sequence, which potentially reduces its overall asymptotic complexity, our technique also allows a user to make a space-time tradeoff by automatically tuning algorithmic and machine parameters using a model-driven framework. Our method improves as the tensor order grows, making its performance more scalable for higher-order data problems. We show speedups of up to $8\times$ and $820\times$ on real sparse data tensors with orders as high as 85 over the SPLATT package and Tensor Toolbox library respectively; and on a full CPD algorithm (CP-ALS), ADATM can be up to $8\times$ faster than state-of-the-art method implemented in SPLATT.**

I. INTRODUCTION

A tensor of order N is an N-way array, which can provide a natural input representation of a multiway dataset. This tensor is sparse if it consists of mostly zero entries.[1] There are several techniques for analyzing and mining a dataset in the tensor form [1–8], which have been applied in a variety of domains, including healthcare [9, 10], natural language processing [11], machine learning [12, 13], and social network analytics [14], among others. In some domains, tensor decomposition methods may be used to compress data [15, 16].

This paper concerns performance enhancement techniques for one of the most popular such methods, the CANDECOMP/PARAFAC decomposition (CPD) [3]. A CPD approximates an input tensor by a sum of component rank-one tensors for a given number of desired components [4, 9, 10, 17, 18]. (It is analogous to computing a truncated singular value decomposition (SVD) of a matrix.) For example, analysts have used the CPD successfully in identifying useful medical concepts, or phenotypes, from raw electronic health records (EHR), which can then be used by medical professionals to facilitate diagnosis and treatment [9, 10]. In this problem example, given a target rank R, CPD provides the top-R patient phenotypes, where each rank-one component of the decomposition corresponds to a phenotype.

The running time of a typical CPD on an N^{th}-order tensor is dominated by the evaluation of a sequence of N *matricized tensor times Khatri-Rao product* (MTTKRP) operations (§ III) [8, 11, 19, 20]. Prior performance studies have focused on optimizing a single MTTKRP [8, 11, 19, 20]. By contrast, we look for ways to improve the *entire* sequence of N MTTKRP*s*, which can lead to a much faster implementation, especially for the higher-order case (i.e., large N). A similar idea has been proposed by Phan et al. [17]; however, our method applies to the sparse (rather than dense) case and exploits the structure of an MTTKRP sequence to preserve sparsity and thereby reduce space.

Contributions. We propose a novel, adaptive tensor memoization algorithm, which we refer to as ADATM.[2] To perform a rank-R CPD of an N^{th}-order sparse tensor with m non-zeros, prior implementations require $\mathcal{O}\big(N^{(1+\epsilon)}mR\big)$ floating-point operations (flops), where $\epsilon \in [0,1]$ is an implementation- and input-dependent parameter [8, 11, 19–21]. By contrast, our proposed method reduces this flop-complexity to $\mathcal{O}\Big(\tilde{N}mR\Big)$, where $\tilde{N}$ is usually much less than $N^{(1+\epsilon)}$; furthermore, the user may control the degree of improvement by trading increased storage for reduced time (smaller $\tilde{N}$) (§ V).

Our method has several parameters, such as tensor features (order, size, non-zero distribution), target rank, and memory capacity. Thus, we develop a model-driven framework to tune them. By model-driven, we mean the framework includes a predictive model for pruning the space candidate implementations, using a user-selectable strategy to prioritize time or space concerns. We further accelerate ADATM within a node by multithreading (§ VI).

ADATM's MTTKRP sequence outperforms the state-of-the-art SPLATT [8] and Tensor Toolbox [22] by up to $8\times$ and $820\times$, respectively, on real sparse tensors with orders as high as 85. Our predictive model effectively selects an optimal implementation. Compared to previous work [8, 11, 19, 20], ADATM scales better as the order (N) grows. For CPD, ADATM achieves up to $8\times$ speedups over SPLATT (§ VII).

II. BACKGROUND

This section introduces some essential tensor notation. Several of its examples and definitions come from the overview

[1]For instance, an $I \times I \times I$ tensor is sparse if its number of non-zero elements, m, satisfies $m \ll I^3$. Indeed, one typically expects $m = \mathcal{O}(I)$.

[2]Read, "Ada-Tee-Em," as in Ada Lovelace.

by Kolda and Bader [3]. A list of symbols and notation in this paper is shown in Table I.

TABLE I
LIST OF SYMBOLS AND NOTATION.

Symbols	Description
$\mathcal{X}, \mathcal{Y}, \mathcal{Y}^{(i)}, \mathcal{Z}^{(i)}$	Sparse or "semi-sparse" tensors
$\mathbf{X}_{(n)}$	Matricized tensor $\mathcal{X}$ in mode-n
$\mathbf{A}, \mathbf{A}^{(i)}, \tilde{\mathbf{A}}^{(i)}$	Dense matrices
$\mathbf{a_r}, \mathbf{b_r}, \mathbf{c_r}, \mathbf{d_r}$	Dense vectors
λ	Weight vector
$\times_n$	Tensor-Times-Matrix multiplication (TTM)
$\otimes$	Kronecker product between two matrices
$\odot$	Khatri-Rao product between two matrices
$*$	Hadamard product between two matrices
$\diamond_n$	quasi Tensor-Times-Matrix multiplication (q-TTM)
N	Tensor order
I, J, K, L, I_i	Tensor mode sizes
m	#Nonzeros of the input tensor $\mathcal{X}$
R	Approximate tensor rank (usually a small value)
m_l	#Fibers at the l^{th}-level of a CSF tensor tree
T_{CP}	Time of CP-ALS
T_M	Time of a single MTTKRP
n_p	#Memoized MTTKRPs in an MTTKRP sequence
m_o	Mode order of a sparse tensor
n_i	#Saved intermediate tensors from a memoized MTTKRP
s	Predicted storage size of ADATM
t	Predicted running time of ADATM
S	Machine space limit

The order N of an N^{th}-order tensor is sometimes also referred to the number of *modes* or *dimensions*. A first-order ($N = 1$) tensor is a vector, which we denote by a boldface lowercase letter, e.g., $\mathbf{v}$; A second-order ($N = 2$) tensor is a matrix, which we denote by a boldface capital letter, e.g., $\mathbf{A}$. Higher-order tensors ($N \geq 3$) are denoted by bold capital calligraphic letters, e.g., $\mathcal{X}$. A scalar element at position (i, j, k) of a tensor $\mathcal{X}$ is x_{ijk}. We show an example of a sparse third-order tensor, $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$, in figure 1(a).

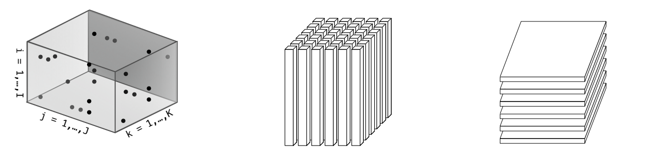
Many tensor algorithms operate on subsets of a tensor. One such subset is the mode-n *fiber*, shown in figure 1(b); it is a vector extracted by fixing the indices of all modes but mode-n. For example, the mode-1 fiber of a tensor $\mathcal{X}$ is denoted by the vector $\mathbf{f}_{:jk} = \mathcal{X}(:, j, k)$, where a colon indicates all indices of the corresponding mode. A *slice*, shown in figure 1(c), is a 2-dimensional cross-section (i.e., matrix) of a tensor, extracted by fixing the indices of all modes but two, e.g $\mathbf{S}_{::k} = \mathcal{X}(:, :, k)$.

A tensor algorithm often reshapes a tensor into an equivalent matrix, a step referred to as *matricization* or *unfolding*. The mode-n matricization of tensor $\mathcal{X}$, denoted by $\mathbf{X}_{(n)}$, arranges all mode-n fibers to be the columns of a matrix. For example, mode-1 matricization of a tensor $\mathcal{X} \in \mathbb{R}^{3 \times 4 \times 5}$ would result in a matrix $\mathbf{X}_{(1)} \in \mathbb{R}^{3 \times 20}$. (Readers may refer to Kolda and Bader's survey for more details [3].)

A. Basic Tensor Operations

In a ***tensor times matrix multiplication (TTM)*** in mode-n, also known as the mode-n product, a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_n \times \cdots \times I_N}$ is multiplied by a matrix $\mathbf{U} \in \mathbb{R}^{R \times I_n}$. This operation is denoted by $\mathcal{Y} = \mathcal{X} \times_n \mathbf{U}$, where $\mathcal{Y} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times R \times I_{n+1} \times \cdots \times I_N}$. In the scalar form, a TTM is

$$y_{i_1 \cdots i_{n-1} r i_{n+1} \cdots i_N} = \sum_{i_n=1}^{I_n} x_{i_1 i_2 \cdots i_N} u_{r i_n}. \quad (1)$$



(a) A third-order sparse tensor (b) Mode-1 fibers: $\mathbf{f}_{:jk} = \mathcal{X}(:, j, k)$. (c) Slices: $\mathbf{S}_{::k} = \mathcal{X}(:, :, k)$.

Fig. 1. Slices and fibers of a third-order tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$, where a colon indicates all indices of a mode.

The ***Kronecker product*** of two matrices $\mathbf{A} \in \mathbb{R}^{I \times K}$ and $\mathbf{B} \in \mathbb{R}^{J \times L}$ is denoted by $\mathbf{C} = \mathbf{A} \otimes \mathbf{B}$, where $\mathbf{C} \in \mathbb{R}^{(IJ) \times (KL)}$. This operation is defined as

$$\mathbf{C} = \mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \dots & a_{1K}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{I1}\mathbf{B} & a_{I2}\mathbf{B} & \dots & a_{IK}\mathbf{B} \end{bmatrix}. \quad (2)$$

The ***Khatri-Rao product*** is a "matching column-wise" Kronecker product between two matrices. Given matrices $\mathbf{A} \in \mathbb{R}^{I \times R}$ and $\mathbf{B} \in \mathbb{R}^{J \times R}$, their Khatri-Rao product is denoted by $\mathbf{C} = \mathbf{A} \odot \mathbf{B}$ where $\mathbf{C} \in \mathbb{R}^{(IJ) \times R}$, or

$$\mathbf{C} = \mathbf{A} \odot \mathbf{B} = [\mathbf{a}_1 \otimes \mathbf{b}_1, \mathbf{a}_2 \otimes \mathbf{b}_2, \dots, \mathbf{a}_R \otimes \mathbf{b}_R], \quad (3)$$

where $\mathbf{a}_r$ and $\mathbf{b}_r$, $r = 1, \dots, R$, are columns of $\mathbf{A}$ and $\mathbf{B}$.

The ***Hadamard product*** $\mathbf{C} = \mathbf{A} * \mathbf{B}$ is the element-wise product between two equal-sized matrices $\mathbf{A}$ and $\mathbf{B}$.

We introduce a simplifying building block, ***quasi tensor times matrix multiplication*** **(q-TTM)** through Hadamard product. Given an $(N+1)^{th}$-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_n \times \cdots \times I_N \times R}$ and a matrix $\mathbf{U} \in \mathbb{R}^{I_n \times R}$, the q-TTM product in mode-n is denoted by $\mathcal{Y} = \mathcal{X} \diamond_n \mathbf{U}$, where $\mathcal{Y} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times I_{n+1} \times \cdots \times I_N \times R}$ is an N^{th}-order tensor. In scalar form, q-TTM is

$$y(i_1, \dots, i_{n-1}, i_{n+1}, \dots, i_N, r) = \sum_{i_n=1}^{I_n} x(i_1, \dots, i_{n-1}, i_n, i_{n+1}, \dots, i_N, r) u(i_n, r). \quad (4)$$

By fixing indices $i_1, \dots, i_{n-1}, i_{n+1}, \dots, i_N$, a Hadamard product is taken between each slice $\mathbf{X}(i_1, \dots, i_{n-1}, :, i_{n+1}, \dots, i_N, :)$ and the matrix $\mathbf{U}$. Every resulting $I_n \times R$ matrix is sum-reduced along with mode-n for all i_n.

The core of the CPD (§ II-C) is the ***matriced tensor times Khatri-Rao product*** **(MTTKRP)**. For an N^{th}-order tensor $\mathcal{X}$ and given matrices $\mathbf{A}^{(1)}, \dots, \mathbf{A}^{(N)}$, the mode-$n$ MTTKRP is

$$\tilde{\mathbf{A}}^{(n)} \leftarrow \mathbf{X}_{(n)} \left(\mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot \cdots \odot \mathbf{A}^{(1)} \right). \quad (5)$$

In the variant of CPD this paper considers, the output $\tilde{\mathbf{A}}^{(n)}$ is used to update the matrix operand $\mathbf{A}^{(n)}$ for the next MTTKRP.

B. Special Properties of Sparse Tensors

There is some additional terminology useful when operating on sparse tensors. When taking a product, TTM or q-TTM, the mode-n in which the product is performed is called the *product mode*; all other modes (e.g., $1, \dots, n-1, n+1, \dots, N$) are the *index modes*. A mode is dense if all its non-empty fibers are dense; otherwise, it is *sparse*. A tensor is called *semi-sparse* if it has one or more dense modes.

A sparse tensor can be efficiently stored in coordinate (COO) ([3]) or Compressed Sparse Fiber (CSF) format ([23]). Both COO and CSF only store the non-zero entries and their indices (in different compressed ways) where tensor product operations (TTM, q-TTM, MTTKRP) perform on.

Some specific properties exist in sparse tensor-dense matrix products TTM and q-TTM. First, sparse TTM outputs a semi-sparse tensor whose product mode is dense and index modes are unchanged. (The details appear elsewhere [6].) Similarly, the q-TTM of a semi-sparse tensor and a dense matrix yields another semi-sparse tensor whose index modes are unchanged while its product mode disappears.

C. *CP Decomposition*

The CANDECOMP/PARAFAC Decomposition (CPD) [3] decomposes a tensor into a sum of component rank-one tensors. For example, CPD of a fourth-order tensor $\mathcal{X} \in \mathbb{R}^{I\times J\times K\times L}$, is approximated as

$$\mathcal{X} \approx [\![\lambda; \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}]\!] \equiv \sum_{r=1}^{R} \lambda_r \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r \circ \mathbf{d}_r, \quad (6)$$

where R is the canonical rank of tensor $\mathcal{X}$, also the number of component rank-one tensors. In a low-rank approximation, R is usually chosen to be a small number less than 100. It is the outer product of the vectors $\mathbf{a}_r, \mathbf{b}_r, \mathbf{c}_r, \mathbf{d}_r$ that produces rank-one tensors, and $\mathbf{A} \in \mathbb{R}^{I\times R}, \mathbf{B} \in \mathbb{R}^{J\times R}, \mathbf{C} \in \mathbb{R}^{K\times R}$, and $\mathbf{D} \in \mathbb{R}^{L\times R}$ are the *factor matrices*, each one formed by taking the corresponding vectors as its columns, i.e., $\mathbf{A} = [\mathbf{a_1}\ \mathbf{a_2}\ ...\ \mathbf{a_r}]$ [3]. We take these vectors to be normalized to have unit magnitude. The vector $\lambda = \{\lambda_1, \ldots, \lambda_r\}$ contains the factor weights.

III. Why Optimize the MTTKRP Sequence?

The most popular algorithm to compute a CPD is arguably the alternating least squares method, or CP-ALS [3, 24]. Each iteration of CP-ALS loops over all modes and updates each mode's factor matrix while keeping the other factors constant. This process repeats until user-specified conditions for convergence or maximum iteration counts are met. An N^{th}-order CP-ALS algorithm appears in algorithm 1. Line 5 is the mode-n MTTKRP. The combined result of all mode-$\{1,\ldots,N\}$ MTTKRPs is the N^{th}-order MTTKRP *sequence*.

Algorithm 1 The N^{th}-order CP-ALS algorithm for a hypercubical tensor.

```
Input: An $N^{th}$-order sparse tensor $\mathcal{X} \in R^{I\times\cdots\times I}$ and an integer rank $R$;
Output: Dense factors $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}$, $\mathbf{A}^{(i)} \in R^{I\times R}$ and weights $\lambda$;
 1: Initialize $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}$;
 2: do
 3:     for $n = 1, \ldots, N$ do
 4:         $\mathbf{V} \leftarrow \mathbf{A}^{(1)\dagger}\mathbf{A}^{(1)} * \cdots \mathbf{A}^{(n-1)\dagger}\mathbf{A}^{(n-1)} *$
                $\mathbf{A}^{(n+1)\dagger}\mathbf{A}^{(n+1)} * \cdots * \mathbf{A}^{(N)\dagger}\mathbf{A}^{(N)}$
 5:         $\tilde{\mathbf{A}}^{(n)} \leftarrow \mathbf{X}_{(n)}(\mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(n+1)} \odot$
                $\mathbf{A}^{(n-1)} \odot \cdots \odot \mathbf{A}^{(1)})$
 6:         $\mathbf{A}^{(n)} \leftarrow \tilde{\mathbf{A}}^{(n)}\mathbf{V}^{\dagger}$
 7:         Normalize columns of $A^{(n)}$ and store the norms as $\lambda$
 8:     end for
 9: while Fit ceases to improve or maximum iterations exhausted.
10: Return: $[\![\lambda, \mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!]$;
```

A. MTTKRP *is the performance bottleneck of CP-ALS.*

Consider an N^{th}-order sparse hypercubical tensor $\mathcal{X} \in R^{I\times\cdots\times I}$ with m nonzeros. The number of flops in *one* iteration of CP-ALS (algorithm 1) is approximately

$$T_{CP} \approx N(N^{\epsilon}mR + NIR^2) \approx NT_M, \quad (7)$$

where $T_M = \mathcal{O}(N^{\epsilon}mR)$ is the time for a single MTTKRP [8, 19, 20, 22]. (The NIR^2 term will typically be neglible because in practice $IR \ll m$.) Tensor Toolbox [19, 22] has $\epsilon = 1$, while SPLATT [8] and DFacTo [20] have $\epsilon \approx 0$ for third-order sparse tensors and $\epsilon \in (0, 1)$ for higher-order sparse tensors.[3] The value of ϵ depends on both the implementation and sparse tensor features, e.g. mode sizes, non-zero distribution. MTTKRPs dominate the running time of CP-ALS: We ran CP-ALS using the state-of-the-art SPLATT library [8] on all of the tensors in table IV (see § VII) and found that MTTKRP accounted for 69-99% of the total running time. Therefore, we focus on optimizing the sequence of MTTKRPs.

B. *The time of an* MTTKRP *sequence grows with tensor order.*

Researchers have successfully optimized a single MTTKRP operation through various methods [8, 11, 19, 20], which reduces the hidden constant of T_M. However, the overall time T_{CP} still grows with the tensor order since CP-ALS executes a *sequence* of N MTTKRPs.

Figure 2 shows the runtime of an MTTKRP sequence using SPLATT on synthetic, hypercubical, sparse tensors generated from Tensor Toolbox [22]. The tensor orders increase from 10 to 80, while the mode size (1,000), rank size R (16) and number of nonzeros (100,000) remain fixed. These synthetic tensors have $\epsilon = 1$, thus the runtime of an MTTKRP sequence increases close to quadratically with N, since each MTTKRP grows with N linearly per equation (7) and the sequence performs N MTTKRPs. Others have improved an MTTKRP sequence by saving large Khatri-Rao product results [17, 25]. In this paper, we propose a new time and space efficient algorithm to solve this problem.

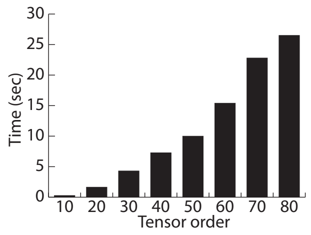


Fig. 2. The runtime of SPLATT sequence on synthetic, sparse tensors.

C. *An* MTTKRP *sequence has arithmetic redundancy.*

In the N^{th}-order MTTKRP sequence in algorithm 1, each mode-n MTTKRP shares all but one factor matrix with the mode-$(n-1)$ MTTKRP. For example, when factoring a fourth-order tensor, the mode-1 MTTKRP $\tilde{\mathbf{A}} \leftarrow \mathcal{X}_{(1)}(\mathbf{D} \odot \mathbf{C} \odot \mathbf{B})$ and the mode-2 MTTKRP $\tilde{\mathbf{B}} \leftarrow \mathcal{X}_{(2)}(\mathbf{D} \odot \mathbf{C} \odot \tilde{\mathbf{A}})$ redundantly compute $\mathbf{D}$ and $\mathbf{C}$ with tensor $\mathcal{X}$. Theoretically, if we could save all of the intermediate results from the mode-1 MTTKRP, we could avoid about $\frac{1}{2}N^2$ redundant computations. The number of redundant computations quadratically increases with the tensor order, which is not scalable. Our proposed method seeks to memoize these redundant computations.

[3]SPLATT and DFacTo improve T_M of a third-order sparse tensor from $3mR$ to $2(m + P)R \approx \mathcal{O}(mR)$ where P is the number of non-empty mode-n fibers, $P \ll m$. However, this $P \ll m$ does not apply when N is large, thus $\epsilon \in (0, 1)$ instead.

IV. MTTKRP Algorithms and Tensor Formats

We introduce two standalone MTTKRP algorithms, which are the building blocks for our memoization scheme. We also separately consider two formats for storing the tensor, one for the sparse case and the other for the semi-sparse case.

A. MTTKRP *Algorithms*

MTTKRP could operate on the nonzeros of a sparse tensor $\mathcal{X}$ without explicitly matricizing. It also avoids explicit Khatri-Rao products, thereby saving space. For example, consider the mode-1 MTTKRP of the fourth-order example in § II-C:

$$\tilde{\mathbf{A}} = \mathbf{X}_{(\mathbf{1})}(\mathbf{D} \odot \mathbf{C} \odot \mathbf{B}), \tag{8}$$

Smith et al. proposed an MTTKRP algorithm, SPLATT [8], which factors out the inner multiplication with $\mathbf{B}$ and $\mathbf{C}$ thereby reducing the number of flops:

$$\tilde{A}(i,r) = \sum_{j=1}^{J} B(j,r) \sum_{k=1}^{K} C(k,r) \sum_{l=1}^{L} \mathcal{X}(i,j,k,l) D(l,r). \tag{9}$$

We define two types of standalone MTTKRP algorithms to help clarify our memoization approach. A *memoized* MTTKRP is the traditional MTTKRP computed from scratch using SPLATT [8] and saving its intermediate results, the semi-sparse tensors (algorithm 2). A *partial* MTTKRP is the MTTKRP computed based on the saved intermediate results from a memoized MTTKRP. It generally takes less time than a memoized MTTKRP. (See its embedded usage in algorithm 3 and 4 in § V.)

Algorithm 2 begins with a TTM, yielding a semi-sparse tensor $\mathcal{Y}^{(1)} \in \mathbb{R}^{I\times J\times K\times R}$, where mode-4 is dense. Then, a q-TTM is performed on $\mathcal{Y}^{(1)}$ with $\mathbf{C}$ to generate the second semi-sparse tensor, $\mathcal{Y}^{(2)}$, with a dense mode-3; then a second q-TTM is performed on $\mathcal{Y}^{(2)}$ with $\mathbf{B}$ to produce the final result $\tilde{\mathbf{A}}$. In this paper, we consider an MTTKRP as the integration of the two *products* TTM and q-TTM. For an arbitrary N^{th}-order tensor, a memoized MTTKRP has $\mathcal{O}(N^{\epsilon} mR)$, $\epsilon \in [0, 1)$ flops, while the number of products is $(N-1)$.

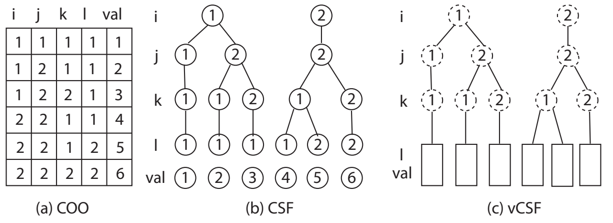
Algorithm 2 Memoized MTTKRP algorithm using SPLATT.

```
Input: A fourth-order sparse tensor $\mathcal{X} \in \mathbb{R}^{I\times J\times K\times L}$, dense factors $\mathbf{B} \in \mathbb{R}^{J\times R}$, $\mathbf{C} \in \mathbb{R}^{K\times R}$, $\mathbf{D} \in \mathbb{R}^{L\times R}$;
Output: Updated dense factor matrix $\tilde{\mathbf{A}} \in R^{I\times R}$;
                                        ▷ $\tilde{\mathbf{A}} \leftarrow \mathcal{X}_{(1)}(\mathbf{D} \odot \mathbf{C} \odot \mathbf{B})$
1: Save $\mathcal{Y}^{(1)}$: $\mathcal{Y}^{(1)} \leftarrow \mathcal{X} \times_4 \mathbf{D}$
2: Save $\mathcal{Y}^{(2)}$: $\mathcal{Y}^{(2)} = \mathcal{Y}^{(1)} \diamond_3 \mathbf{C}$
3: $\tilde{\mathbf{A}} = \mathcal{Y}^{(2)} \diamond_2 \mathbf{B}$
4: return $\tilde{\mathbf{A}}$;
```

B. *CSF and vCSF Formats*

Besides the algorithms, we need to specify a data structure for the sparse tensors. The most commonly used format is the coordinate (COO) format of figure 3(a). More recently, Smith et al. proposed a Compressed Sparse Fiber (CSF) format, which is a hierarchical, fiber-centric format that effectively extends the compressed sparse row (CSR) format of sparse matrices to sparse tensors, illustrated in figure 3(b) [23]. CSF is memory-efficient and generally leads to a faster MTTKRP compared to COO [8, 23]. Therefore, our tensor memoization algorithm assumes CSF for storing the input sparse tensor; to store the intermediate semi-sparse tensors, we use a variant of CSF, which we call vCSF, shown in figure 3(c).

In CSF, each root-to-leaf path corresponds with a coordinate tuple of a nonzero entry, where replicated indices at the same level are compressed to reduce storage. When computing a sparse tensor-dense matrix product (TTM), the output tensor $\mathcal{Y}^{(1)}$ in algorithm 2 has the same i, j, k indices with $\mathcal{X}$ and a dense mode-l (see § II-B). Thus, when storing a semi-sparse tensor $\mathcal{Y}^{(1)}$, we do not need to store indices of all modes since i, j, k can be reused from $\mathcal{X}$ and dense indices l need not be stored. We refer to this storage scheme as vCSF, as an auxiliary format of CSF, storing only the nonzero values of an intermediate semi-sparse tensor.



i	j	k	l	val
1	1	1	1	1
1	2	1	1	2
1	2	2	1	3
2	2	1	1	4
2	2	1	2	5
2	2	2	2	6

Fig. 3. A sparse tensor $\mathcal{X} \in \mathbb{R}^{2\times 2\times 2\times 2}$ in COO and CSF formats, tensor $\mathcal{Y}^{(1)}$ in vCSF format without storing indices.

V. Memoization for the MTTKRP Sequence

We present two tensor memoization algorithms for the MTTKRP sequence. To simplify the presentation, we show them assuming the fourth-order example of § II-C but have implemented the general algorithm for N^{th}-order tensors.

A. *Two Fourth-Order Tensor Memoization Algorithms*

1) A Basic Memoization Scheme: A simple memoized algorithm appears in algorithm 3, shown for simplicity for the fourth-order case ($N = 4$). It saves every intermediate semi-sparse tensor ($\mathcal{Y}^{(1)}$ and $\mathcal{Y}^{(2)}$) generated from the mode-1 MTTKRP using the memoized MTTKRP algorithm (algorithm 2) and then reuses them to speedup subsequent MTTKRPs using the partial MTTKRP algorithm. Note that the mode-3 MTTKRP has more flops than the mode-2 one. The mode-4 MTTKRP cannot reuse any saved intermediates, therefore, it is computed from scratch using SPLATT [8], which directly operates on the tensor $\mathcal{X}$ and does not save any intermediate tensors. Figure 4(b) illustrates this algorithm.

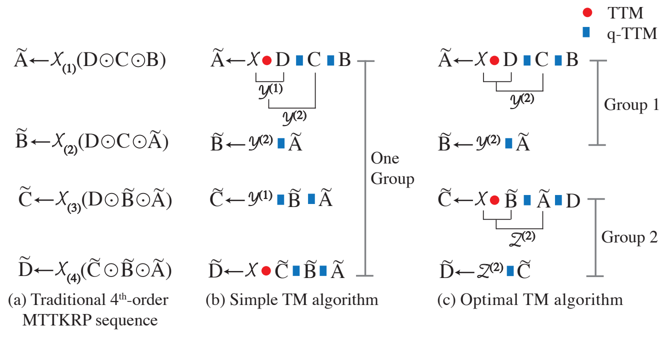


Fig. 4. Graphical process of the simple and optimal tensor memoization algorithms. "red circle" represents TTM and "blue block" is q-TTM.

Algorithm 3 A simple tensor memoization algorithm of a fourth-order sparse MTTKRP sequence.

```
Input: A fourth-order sparse tensor $\mathcal{X} \in R^{I\times J\times K\times L}$, dense initial factors
    $\mathbf{A} \in R^{I\times R}$, $\mathbf{B} \in R^{J\times R}$, $\mathbf{C} \in R^{K\times R}$, $\mathbf{D} \in R^{L\times R}$;
Output: Updated factors $\tilde{\mathbf{A}}, \tilde{\mathbf{B}}, \tilde{\mathbf{C}}, \tilde{\mathbf{D}}$;
                                        $\triangleright\ \tilde{\mathbf{A}} \leftarrow \mathcal{X}_{(1)}(\mathbf{D}\odot\mathbf{C}\odot\mathbf{B})$.
1: Save $\mathcal{Y}^{(1)}$: $\mathcal{Y}^{(1)} \leftarrow \mathcal{X} \times_4 \mathbf{D}$;
2: Save $\mathcal{Y}^{(2)}$: $\mathcal{Y}^{(2)} \leftarrow \mathcal{Y}^{(1)} \diamond_3 \mathbf{C}$;
3: $\tilde{\mathbf{A}} \leftarrow \mathcal{Y}^{(2)} \diamond_2 \mathbf{B}$;
                                        $\triangleright\ \tilde{\mathbf{B}} \leftarrow \mathcal{X}_{(2)}(\mathbf{D}\odot\mathbf{C}\odot\tilde{\mathbf{A}})$.
4: Reuse $\mathcal{Y}^{(2)}$: $\tilde{\mathbf{B}} \leftarrow \mathcal{Y}^{(2)} \diamond_1 \tilde{\mathbf{A}}$
                                        $\triangleright\ \tilde{\mathbf{C}} \leftarrow \mathcal{X}_{(3)}(\mathbf{D}\odot\tilde{\mathbf{B}}\odot\tilde{\mathbf{A}})$.
5: Reuse $\mathcal{Y}^{(1)}$: $\tilde{\mathbf{C}} \leftarrow (\mathcal{Y}^{(1)} \diamond_2 \tilde{\mathbf{B}}) \diamond_1 \tilde{\mathbf{A}}$
                                        $\triangleright\ \tilde{\mathbf{D}} \leftarrow \mathcal{X}_{(4)}(\tilde{\mathbf{C}}\odot\tilde{\mathbf{B}}\odot\tilde{\mathbf{A}})$.
6: SPLATT: $\tilde{\mathbf{D}} \leftarrow \mathcal{X}_{(4)}(\tilde{\mathbf{C}}\odot\tilde{\mathbf{B}}\odot\tilde{\mathbf{A}})$
7: Return: $\tilde{\mathbf{A}}, \tilde{\mathbf{B}}, \tilde{\mathbf{C}}, \tilde{\mathbf{D}}$;
```

2) An Optimal Memoized Scheme: The simple method of algorithm 3, which memoizes all intermediates from one MTTKRP in sequence, is not the only approach.

For instance, consider algorithm 4. Whereas algorithm 3 memoizes only the mode-1 MTTKRP, algorithm 4 memoizes with respect to the mode-1 and mode-3 MTTKRPs. In particular, it saves only one intermediate tensor $\mathcal{Y}^{(2)}$ during the mode-1 MTTKRP, which is then reused in the mode-2 MTTKRP; then, during the mode-3 MTTKRP it saves $\mathcal{Z}^{(2)}$. To be able to apply $\mathcal{Z}^{(2)}$ during the mode-4 MTTKRP, we need to permute $\mathcal{X}$, which in our scheme creates another sparse tensor $\tilde{\mathcal{X}} \in \mathbb{R}^{K\times J\times I\times L}$. (It is possible to avoid this extra storage cost but we do not consider that in this paper to pursue better performance.) Figure 4(c) illustrates this algorithm.

Algorithm 4 An optimal tensor memoization algorithm of a fourth-order sparse MTTKRP sequence.

```
Input: A fourth-order sparse tensor $\mathcal{X} \in R^{I\times J\times K\times L}$, dense initial factors
    $\mathbf{A} \in R^{I\times R}$, $\mathbf{B} \in R^{J\times R}$, $\mathbf{C} \in R^{K\times R}$, $\mathbf{D} \in R^{L\times R}$;
Output: Updated factors $\tilde{\mathbf{A}}, \tilde{\mathbf{B}}, \tilde{\mathbf{C}}, \tilde{\mathbf{D}}$;
                                        $\triangleright\ \tilde{\mathbf{A}} \leftarrow \mathcal{X}_{(1)}(\mathbf{D}\odot\mathbf{C}\odot\mathbf{B})$.
1: Save $\mathcal{Y}^{(2)}$: $\mathcal{Y}^{(2)} \leftarrow (\mathcal{X} \times_4 \mathbf{D}) \diamond_3 \mathbf{C}$;
2: $\tilde{\mathbf{A}} \leftarrow \mathcal{Y}^{(2)} \diamond_2 \mathbf{B}$;
                                        $\triangleright\ \tilde{\mathbf{B}} \leftarrow \mathcal{X}_{(2)}(\mathbf{D}\odot\mathbf{C}\odot\tilde{\mathbf{A}})$.
3: Reuse $\mathcal{Y}^{(2)}$: $\tilde{\mathbf{B}} \leftarrow \mathcal{Y}^{(2)} \diamond_1 \tilde{\mathbf{A}}$
                                        $\triangleright\ \tilde{\mathbf{C}} \leftarrow \mathcal{X}_{(3)}(\tilde{\mathbf{B}}\odot\tilde{\mathbf{A}}\odot\mathbf{D})$.
   // Permute $\mathcal{X}$ to $\tilde{\mathcal{X}} \in \mathbb{R}^{K\times L\times I\times J}$.
4: Save $\mathcal{Z}^{(2)}$: $\mathcal{Z}^{(2)} \leftarrow (\tilde{\mathcal{X}} \times_4 \tilde{\mathbf{B}}) \diamond_3 \tilde{\mathbf{A}}$;
5: $\tilde{\mathbf{C}} \leftarrow \mathcal{Z}^{(2)} \diamond_2 \mathbf{D}$;
                                        $\triangleright\ \tilde{\mathbf{D}} \leftarrow \mathcal{X}_{(4)}(\tilde{\mathbf{B}}\odot\tilde{\mathbf{A}}\odot\tilde{\mathbf{C}})$.
6: Reuse $\mathcal{Z}^{(2)}$: $\tilde{\mathbf{D}} \leftarrow \mathcal{Z}^{(2)} \diamond_1 \tilde{\mathbf{C}}$
7: Return: $\tilde{\mathbf{A}}, \tilde{\mathbf{B}}, \tilde{\mathbf{C}}, \tilde{\mathbf{D}}$;
```

Algorithm 4 needs 8 products counting both TTM and q-TTM products, which is one less than algorithm 3. However, the permuted sparse tensor $\tilde{\mathcal{X}}$ takes extra space.

For an N^{th}-order tensor, algorithm 3 only memoizes the mode-1 MTTKRP, and its total number of products of the MTTKRP sequence grows like $\mathcal{O}(N^2)$; by contrast, algorithm 4 memoizes more MTTKRPs, reducing the number of products to $\mathcal{O}(N^{1.5})$ (See § V-B). That is, this reduction is *asymptotic* in N. As it happens, this choice of memoization strategies is optimal under certain conditions. However, it also requires extra space to store intermediate tensors and the permuted tensor. Thus, choosing a strategy may require trading space for time. This motivates our proposed scheme, which tries to find a time-optimal strategy guided by a user-specified space limit.

B. Memoization Strategy Analysis

To see how the choice of memoization strategy can lead to asymptotic reductions in operation count, consider a simplified analysis for the following problem:

> **Problem**: *Find the number of memoized* MTTKRP*s that minimizes the total number of products (*TTM *and q-*TTM*) in an N^{th}-order* MTTKRP *sequence.*

Though our experiments involve general sparse tensors, for this analysis suppose the input tensor $\mathcal{X} \in \mathbb{R}^{I\times\cdots\times I}$ is hypercubical and dense. Since each product may consume different amounts of flops, we assume an average number of flops per product that is fixed for a given tensor $\mathcal{X}$. We also assume each partial MTTKRP reuses as many of the memoized intermediate tensors as possible.

A *producer mode* is one that uses a memoized MTTKRP and produces intermediate tensors, while a *consumer mode* uses a partial MTTKRP that consumes these intermediates. One producer mode and some consumer modes constitute an MTTKRP *group*, in which all the consumer modes reuse the intermediate tensors that the producer mode memoized. Let the number of producer modes and consumer modes be n_p and n_c, respectively, where $n_c = N - n_p$; and let the total number of products in the N^{th}-order MTTKRP sequence be n_O. Thus, $n_p = 1$ and $n_O = \mathcal{O}(N^2)$ in algorithm 3 and $n_p = 2$ and $n_O = \mathcal{O}(N^{1.5})$ in algorithm 4.

Lemma 5.1: Given n_p producer modes, the minimum n_O can be achieved by evenly assigning producer modes to consumer modes for reuse.

Proof: Consider any assignment of c_i consumer modes to the i^{th} producer mode in group-i, where $i = 1, ..., n_p$. Then,

$$n_p + n_c = n_p + \sum_{i=1}^{n_p} c_i = N. \tag{10}$$

The products in group-i consists of $(N-1)$ products in the one memoized MTTKRP plus the products of c_i partial MTTKRPs. From figure 4(b), the number of products of all c_i consumer modes in group-i is $1 + \cdots + c_i = c_i(c_i+1)/2$; thus the number of products in group-i,

$$O_i = N - 1 + \frac{c_i(c_i+1)}{2}. \tag{11}$$

The total number of products of an MTTKRP sequence is

$$n_O = \sum_{i=1}^{n_p} O_i = \frac{1}{2}\left[\sum_{i=1}^{n_p} c_i^2 + \sum_{i=1}^{n_p} c_i + 2n_p(N-1)\right]. \tag{12}$$

Substituting equation (10) yields

$$n_O = \frac{1}{2}\left[\sum_{i=1}^{n_p} c_i^2 + 2n_pN + N - 3n_p\right]. \tag{13}$$

Since n_p and N are fixed, minimizing n_O is equivalent to minimizing the sum of squares, $\sum_{i=1}^{n_p} c_i^2$, subject to the constraint equation (10). This elementary optimization problem has a minimum when $c_1 = \cdots = c_{n_p}$, by applying the Cauchy-Schwarz inequality. Therefore, the n_p producer modes should be evenly assigned to approximately $\frac{n_c}{n_p}$ consumer modes for reuse in order to minimize n_O. ■

Lemma 5.2: $n_p^* = \sqrt{N/2}$ minimizes n_O for an N^{th}-order MTTKRP sequence.

Proof: For simplicity, assume n_p divides N. Then, by applying lemma 5.1,

$$c_i = N/n_p - 1, i = 1, \ldots, n_p. \tag{14}$$

Substituting equation (14) into equation (13),

$$n_O = \frac{1}{2}\left[\frac{N^2}{n_p} + 2(N-1)n_p - N\right], 1 \leq n_p \leq \frac{N}{2}, \tag{15}$$

where the largest possible $\frac{N}{2}$ is choosing all n_p by memoizing every other mode.

By taking the derivative of n_O with respect to n_p, the minimum occurs at

$$n_p^* = \frac{N}{\sqrt{2(N-1)}} \approx \sqrt{\frac{N}{2}}. \tag{16}$$

The minimum n_O is $n_O^* \approx \sqrt{2}N^{1.5}$. ■

We know $n_O = N(N+1)/2$ when $n_p = 1$ and $n_O = N^2/2$ when $n_p = N/2$. The optimal n_p^* achieves the minimum $n_O^* = \mathcal{O}(N^{1.5})$, which is asymptotically better especially for higher-order tensors. Lemma 5.2 shows that even if we have infinite storage space to memoize all $N/2$ MTTKRPs, that would not actually minimize the number of products. Intuitively, even though consumer modes benefit from the saved intermediate tensors, the memoized MTTKRP itself consumes $(N-1)$ operations. This analysis shows there is an optimal balance.

Given an input tensor, we first calculate the optimal number of memoized MTTKRPs based on these lemmas, and then design our adaptive tensor memoization algorithm (ADATM) such as algorithm 4 for fourth-order tensors. As we assumed, this optimal n_p is calculated on dense, hypercubical tensors, it is not accurate for diverse sparse tensors. In § VI, we construct a model-driven framework to predict the optimal parameters considering input tensor features and storage efficiency.

VI. ADATM: ADAPTIVE TENSOR MEMOIZATION

Based on the discussion of § V, we are motivated to develop a framework to choose an optimal memoization strategy. This section explains our approach, which we refer to as the adaptive tensor memoization framework, ADATM.

A. Parameter Selection

The memoized algorithm has several natural parameters, which can be tuned to trade storage for time:

1) n_p: The number of producer modes (or memoized MTTKRPs). Lemma 5.2 shows that, in theory, there exists an optimal choice for n_p; for hypercubical dense tensors, $n_p^* = \sqrt{N/2}$. Therefore, our adaptive framework heuristically considers all $n_p \in \{1, \ldots, \sqrt{N/2}\}$. (In our experiments, we find that the optimal n_p is always smaller than n_p^*.)

2) m_o: The mode order of a sparse tensor. As the discussion of algorithm 4 suggests, one may choose the order of modes for each memoized MTTKRP. The two memoized MTTKRPs in algorithm 4 are $\mathcal{X} \in \mathbb{R}^{I\times J\times K\times L}$ and $\tilde{\mathcal{X}} \in \mathbb{R}^{K\times L\times I\times J}$, with different mode orders in each case. The new $\tilde{\mathcal{X}} \in \mathbb{R}^{K\times L\times I\times J}$ is created that mode-4 MTTKRP is able to reuse the memoized semi-sparse tensor $\mathcal{Z}^{(2)} \in \mathbb{R}^{K\times L\times R}$. In fact, $\hat{\mathcal{X}} \in \mathbb{R}^{K\times L\times J\times I}$ can also generate the same $\mathcal{Z}^{(2)}$, so we heuristically choose to contract long modes first, as originally suggested in the SPLATT work [8]. Thus, if $J > I$, we prefer $\tilde{\mathcal{X}} \in \mathbb{R}^{K\times L\times I\times J}$.

3) n_i: The number of intermediate semi-sparse tensors saved from each memoized MTTKRP. For each memoized MTTKRP, the choice of n_i allows us to trade space for time. The range of n_i is $\{1, \ldots, N/n_p - 1\}$.

For any choice of preceding parameters, we have a model that estimates the storage $s(n_p, m_o, n_i)$ in bytes and time $t(n_p, m_o, n_i)$ in flops (see below). This capability is what enables us to quickly search the parameter space. Besides, armed with this model, we are able to construct a model-driven framework to find good values of these parameters.

B. A Model-Driven Framework

Our model-driven framework tunes the parameters (n_p, m_o, n_i) and then selects an ADATM implementation, as illustrated in figure 5. The inputs are the order N of a sparse tensor, its number of nonzeros m, and the target rank R. The user may specify a memory limit and preferred strategy, e.g., maximize performance or minimize storage by a certain degree. The framework considers a large set of configurations, then uses a predictive model to estimate each configuration's storage and time and prune the candidates that, for instance, do not meet the memory limit.

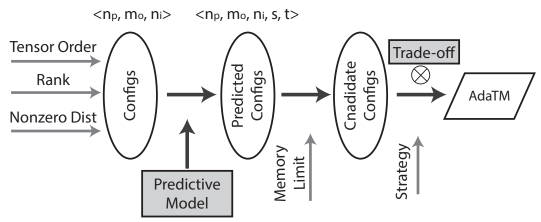


Fig. 5. The model-driven framework.

C. Predictive Model

The input sparse tensor is stored in the standard coordinate format, we predict the space s as the sum of n_p sparse tensors in CSF format and $n_p \times n_i$ semi-sparse tensors in vCSF format both under the mode order m_o. The time t is predicted by adding the flops of all products, TTMs and q-TTMs. The model for t need not be accurate in an absolute sense; rather, it need only be accurate enough to produce a correct relative ranking.

We show the formulas used for the estimates of s and t in table II. The number of fibers at the l^{th}-level of a CSF tree is m_l for $l = 1, \ldots, N$ in figure 3(b), where $m_1 \leq \cdots \leq m_N = m$. (See [23] for details.) We assume the sparse and semi-sparse tensors in evaluating TTM and q-TTM products are both in order-N and have m nonzeros. And since a CSF tensor is stored hierarchically, $m_{CSF} = 16\sum_{l=1}^{N} m_l$. In q-TTM, the semi-sparse tensor in vCSF format only stores nonzero values, $8m$ bytes. One MTTKRP group consists of one memoized MTTKRP and $(N/n_p - 1)$ partial MTTKRPs. The memoized MTTKRP takes $2\sum_{l=2}^{N} m_l R = \mathcal{O}(N^{\epsilon} mR)$ time, $\epsilon \in [0, 1)$, since $m_l \leq m$, when $l < N$. The partial MTTKRPs in the same group, adhered to this memoized MTTKRP, have different nonzeros for each semi-sparse tensor. Finally, we add all n_p MTTKRP groups up and get the time and space of ADATM algorithm for an MTTKRP sequence. When using SPLATT [8] to compute an MTTKRP sequence, the space is smaller than ADATM with the price of slower execution. From this table we get

$$s = \sum_{i=1}^{n_p} \left(m_{CSF}^i + 8 \sum_{l=\frac{N}{n_p}-n_i+1}^{\frac{N}{n_p}} m_l R \right) \tag{17}$$

$$t = 2\sum_{i=1}^{n_p} \left(\sum_{l=2}^{N} m_l R + \sum_{l=1}^{\frac{N}{n_p}-1} \sum_{j=2}^{l+1} m_j \right) R \triangleq 2\tilde{N} m R. \tag{18}$$

D. Strategy Guided Trade-off

Having predicted time and space, the framework searches the implementation under two strategy options: *performance-essential*, for the maximum performance; *space-efficient*, for the close-to-maximum performance but saving more space. In particular, for space-efficient strategy ADATM chooses the configuration with the smallest predicted space while having predicted performance no worse than some fraction of the optimal performance of all candidate configurations. (Our experiments use a fraction of 90%.)

E. Parallelization

Two parallel strategies are employed in ADATM: parallelizing among one mode and among multiple modes. Multiple-mode parallelism becomes possible because a partial MTTKRP can be parallelized among all the fibers of the saved intermediate tensors. For lower-order tensors with long modes, this strategy may not be attractive because parallelizing only one mode can expose enough parallelism for relatively small numbers of cores, as for current multi-core architectures. Since multiple-mode parallelism is finer-grained than single-mode parallelism, for higher-order tensors with short modes, this strategy has performance advantages. Additionally, it would most likely map better to manycore co-processors (e.g., NVIDIA GPUs, Intel Xeon Phi), though we have not explored this possibility in this paper.

VII. EXPERIMENTS AND ANALYSES

Our experimental evaluation considers (a) reporting the speedup of ADATM over the state-of-the-art SPLATT and Tensor Toolbox libraries; (b) analyzing the needed storage to obtain this speedup; (c) evaluating the space-time tradeoff to choose the optimal n_p; (d) assessing thread scalability and dimension scalability; and (e) verifying our model. Lastly, we apply our framework to CPD and show its performance.

A. Data Sets and Platforms

This evaluation uses the two platforms shown in table III. We mainly show the results on the Xeon-based system because of its larger memory size and more cores, while the Core-based system is used to verify the portability of our model. All computations are performed in double-precision and the rank R is set to 16.

TABLE III
EXPERIMENTAL PLATFORMS CONFIGURATION

Parameters	Intel Xeon E7-4820	Intel Core i7-4770K
Microarchitecture	Westmere	Haswell
Frequency	2.0 GHz	3.5 GHz
# of physical cores	16	4
Memory size	512 GiB	32 GiB
Memory bandwidth	34.2 GB/s	25.6 GB/s
Compiler	gcc 4.4.7	gcc 4.7.3

We use two types of data. The first are third-order sparse tensors from real applications including Never Ending Language Learning (NELL) project [26] ("nell1" and "nell2" with *noun-verb-noun*) and data crawled from tagging systems [27] ("deli" with *user-item-tag*). (Refer FROSTT [28], the Formidable Repository of Open Sparse Tensors and Tools, for more details.) The second are higher-order, hyper-sparse tensors constructed from Electronic Health Records (EHR), with orders as high as 85. (In domains other than data analysis, constructing higher-order tensors to do low-rank approximation has been extensively used to achieve good data compression [15, 16].) Additional details appear in table IV.

TABLE IV
DESCRIPTION OF SPARSE TENSORS.

Dataset	Order	Max Mode Size	NNZ	Density
nell2	3	30K	77M	1.3e-05
nell1	3	25M	144M	3.1e-13
deli	3	17M	140M	6.1e-12
ehr36	36	19	11K	4.7e-26
ehr71	71	21	221K	1.4e-55
ehr85	85	21	920K	7.9e-68

B. Performance

We test the speedups of ADATM over SPLATT [29] and Tensor Toolbox libraries [22] for an MTTKRP sequence in figure 6.[4] SPLATT v1.1.1 is tested under "SPLATT_CSF_TWOMODE" mode, which is usually the best case, without preprocessing nor tiling [5] , and Tensor Toolbox v2.6 is tested in MATLAB R2014a environment. ADATM achieves a speedup up to $1.7\times$ for the third-order tensors and up to $8.0\times$ for the higher-order tensors compared to the best parallel performance of SPLATT. The higher speedups on higher-order tensors in figure 6(a) show the advantage of ADATM with respect to the tensor order. Tensor *deli* merely show speedup since ADATM in mode-2 has worse threading scalability than SPLATT. Comparing to Tensor Toolbox, ADATM obtains $93-820\times$ speedups, because Tensor Toolbox uses COO format and a less efficient and sequential MTTKRP algorithm. Tensor *nell2* achieves an even higher speedup than higher-order tensors in figure 6(b) because of its best threading scalability among all tensors. (Its sequential speedup over Tensor Toolbox ($58\times$) is much less than that of higher-order tensors ($325-617\times$).) We mainly use SPLATT to evaluate our algorithm below.

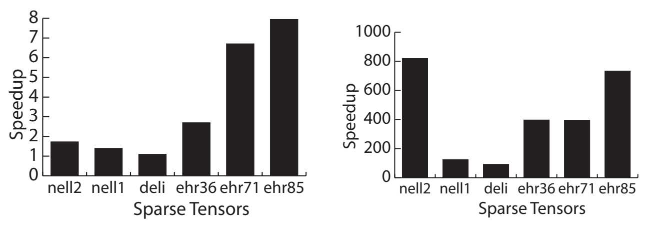
[4]We show speedups because the running time of the MTTKRP sequence varies a lot on these tensors.

[5]Cache tiling is beneficial cooperatively with tensor reordering in [8], which requires expensive pre-processing.

TABLE II
THE NUMBER OF FLOPS AND STORAGE SIZE OF PRODUCTS AND ALGORITHMS.

Algorithms		#Flops	Storage Size (Bytes)
Product	TTM	$2mR$	m_{CSF}
	q-TTM	$2mR$	$8m$
One MTTKRP group	Memoized MTTKRP	$2\sum_{l=2}^{N} m_l R$	$m_{CSF} + 8\sum_{l=\frac{N}{n_p}-n_i+1}^{\frac{N}{n_p}} m_l R$
	Partial MTTKRPs	$2\sum_{l=1}^{\frac{N}{n_p}-1} \sum_{j=2}^{l+1} m_j R$	-
MTTKRP sequence	ADATM	$2\sum_{i=1}^{n_p} \left(\sum_{l=2}^{N} m_l R + \sum_{l=1}^{\frac{N}{n_p}-1} \sum_{j=2}^{l+1} m_j \right) R$	$\sum_{i=1}^{n_p} \left(m_{CSF}^{i} + 8\sum_{l=\frac{N}{n_p}-n_i+1}^{\frac{N}{n_p}} m_l R \right)$
	SPLATT [8]	$2NmR$	m_{CSF}

Indices and values use "uint64_t" and "double" respectively. m_l is the number of fibers at the l^{th}-level of a CSF tree in figure 3(b), $m_{CSF} = 16\sum_{l=1}^{N} m_l$.



(a) SPLATT (b) Tensor Toolbox

Fig. 6. Speedup of ADATM over SPLATT and Tensor Toolbox.

C. Analysis

1) Storage: Table V shows the needed storage for all tensors, when achieving the best performance in figure 6. Traditional software (e.g. Tensor Toolbox) uses COO format, while SPLATT proposed the more efficient CSF format. Since we use "SPLATT_CSF_TWOMODE" mode for good SPLATT performance, two CSF tensors are created (refer to [23] for details). Thus, some tensors in CSF format need more space than in COO format. ADATM's space is shown as "CSF+vCSF" by summing the space of CSF format for sparse tensors and vCSF format for semi-sparse tensors. We calculate the ratios of the ADATM's space to COO and CSF respectively in the right-most two columns. ADATM uses 102%-411% space of CSF and 78%-265% of COO, while achieving up to 8× speedup in figure 6. We also calculated the space for tensor *nell2* using the "avoid duplicated computation" method in [25], which stores the large dense matrices generated from Khatri-Rao product chain, 77952 MBytes extra space is needed. This is much larger than the storage size of ADATM (2581 MBytes). When ADATM is guided by "space-efficient" strategy, we can save some space without harming much performance.

TABLE V
STORAGE SIZE OF SPARSE TENSORS.

	Storage Size (MBytes)			Ratios	
Dataset	COO	CSF	CSF+vCSF	/CSF	/COO
nell2	2290	2540	2581	102%	113%
nell1	4280	6430	8510	132%	199%
deli	4180	5570	11090	199%	265%
ehr36	3.04	1.94	7.97	411%	262%
ehr71	121	62	205	333%	169%
ehr85	604	200	470	236%	78%

2) Choosing n_p: Theoretic optimal $n_p^* = 7$ is calculated from lemma 5.2 for tensor *ehr85*, figure 7 shows the relation between time and space for n_p in the range $[1, 7]$. $n_p = 0$ is the time and space of SPLATT. ADATM gets the shortest execution time on $n_p = 2$, when $n_p > 2$ the running time of ADATM increases as well as its space. Because of this relation, ADATM can adapt the output configuration according to user-defined tradeoff strategy. Though the space on $n_p = 2$ is 236% of SPLATT's, ADATM achieves 6.4× speedup.

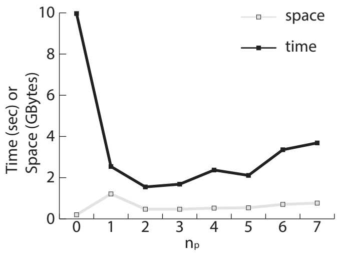
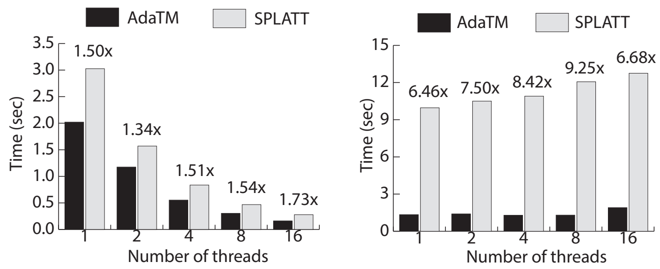


Fig. 7. Time and space relation for *ehr85*.

3) Scalability: We analyze thread scalability and dimension scalability. Figure 8 shows the thread scalability of ADATM and SPLATT on tensors *nell2* and *ehr85*. The numbers above the bars are the speedup of ADATM over SPLATT when using the same number of threads. ADATM and SPLATT both obtain good scalability on *nell2*. However, SPLATT doesn't scale at all on *ehr85*, while ADATM gets the highest performance on 4 threads. The speedups in the two figures mainly increase with growing thread numbers, showing ADATM has better scalability using multiple-mode parallelism. Similar behavior is also observed from the other tensors, demonstrating that the thread scalability of higher-order tensors need further improvement.



(a) *nell2* (b) *ehr85*

Fig. 8. Multithreading scalability of ADATM and SPLATT on *nell2* and *ehr85*.

To better evaluate the dimension scalability, we use the method from [30] in Tensor Toolbox to create synthetic higher-order sparse tensors. We generate eight sparse hypercubical tensors from 10^{th} to 80^{th}-order, all with 100,000 nonzeros and equal mode size 1000. The running time of ADATM and SPLATT is shown in figure 9. As tensor order grows, SPLATT's time is increasing much faster than ADATM's time, which verifies ADATM's good speedup on higher-order tensors.

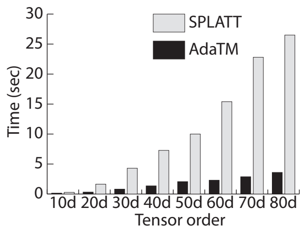


Fig. 9. ADATM's dimension scalability on synthetic sparse tensors.

4) Model Analysis: We test all possible n_p of *ehr85* to record their actual time on the two platforms and compare with our predicted time in § VI. Figure 10 draws the relative value by normalizing to each one's time value on $n_p = 1$. Model-predicted time can find the optimal n_p, which is 2, and predicts a similar trend to the actual time on the two platforms. Since the MTTKRP is dominated by TTM and q-TTM products which have relatively predictable behavior by timing a dense matrix, our model works well. However, since our model hasn't considered architecture characteristics, the prediction is constant for different platforms.

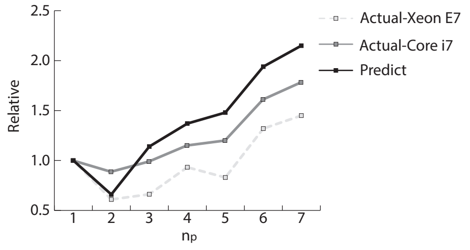


Fig. 10. Accuracy of ADATM model on Xeon E7-4820 and Core i7-4770K.

D. Application

We compare the running time of CP-ALS using ADATM and SPLATT in figure 11. Since the MTTKRP sequence dominates CP-ALS, ADATM shows good speedups especially on higher-order tensors. ADATM speedups CP-ALS by $4\%-69\%$ for the third-order tensors and $2-8\times$ for the higher-order tensors. The speedups are similar to or slightly lower than those of MTTKRP sequence in figure 6(a). This figure shows ADATM is applicable to tensor decompositions in real applications.

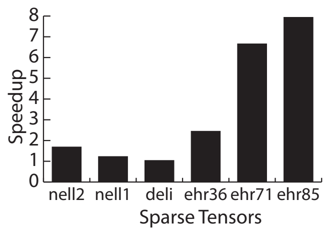


Fig. 11. CP-ALS runtime using SPLATT and ADATM.

VIII. Related Work

Researchers have successfully optimized a single MTTKRP operation through various methods. Tensor Toolbox [22] and Tensorlab [31] implemented MTTKRP in the most popular COO format by multiple sparse tensor-vector products using MATLAB, with a high number of flops $\mathcal{O}(NmR)$. SPLATT [8, 23], the implementation achieving the highest performance so far, algorithmically improved MTTKRP by factoring out inner multiplications and proposed a more compressed CSF format, reducing MTTKRP to $\mathcal{O}(N^{\epsilon}mR), \epsilon \in [0,1)$ flops. GigaTensor [11] reformulated MTTKRP as a series of Hadamard products to utilize the massive parallelism of MapReduce. However, this algorithm is not suitable for multi-core machines because of its high complexity ($\mathcal{O}(5mR)$ for third-order tensors). DFacTo [20] considered MTTKRP as a series of sparse matrix-vector multiplications for distributed systems. Though it has the same number of flops as SPLATT, the explicit matricization takes non-negligible time. HyperTensor [19] investigated fine- and coarse-grained distributed algorithms also for distributed systems, while its MTTKRP implementation is based on COO format. These approaches require $\mathcal{O}\left(N^{(1+\epsilon)}mR\right), \epsilon \in [0,1]$ flops for the MTTKRP sequence. Our work is based on the SPLATT algorithm by memoizing intermediate results to explore data reuse and reduces this flop-complexity to $\mathcal{O}\left(\tilde{N}mR\right)$, where $\tilde{N}$ is generally much less than $N^{(1+\epsilon)}$.

Recently, S. Zhou et al. [25] and A. Phan et al. [17] exploited data reuse for the MTTKRP sequence by storing the results of dense Khatri-Rao products. The extra storage is $\Theta\left(I^{(N-1)}R\right)$ and $\Omega\left(I^{\frac{N}{2}}R\right)$ respectively for a hypercubical sparse tensor, which is extremely larger than the number of nonzeros and exponentially grows with tensor order N. ADATM efficiently stores intermediate semi-sparse tensors instead and in a very compressed pattern to avoid redundant computation and also memory blowup. Baskaran et al. [32] and Kaya et al. [33] also brought similar ideas to save intermediate results in Tucker decomposition. However, our work does a detailed analysis on the most significant factor – the space and time tradeoff. Very recently, Kaya et al. [34] applied the above method in CP decomposition by using dimension trees and considered MTTKRP as a group of tensor-times-vector products. Our work uses TTM and q-TTM for better data locality, and ADATM allows user-defined strategy for the space-time tuning in our model-driven framework.

IX. Conclusion and Future Work

The methods underlying ADATM derive performance improvements from three key ideas. First, we consider not just a single MTTKRP, but the sequence as it arises in the context of CPD. Indeed, the lemmas of § V and optimization method apply to other algorithms that might involve a Khatri-Rao product chain, such as the CP-APR algorithm [30]. Secondly, we develop an "any-space" memoization technique that permits a gradual tradeoff of storage for time. A user or higher-level library may therefore control our method according to whichever criterion is more important in a given application. Thirdly, we parameterize our algorithm and build a model-driven and user-guided framework for it. This technique gives end-user application developers some flexibility in how they employ our algorithm.

Looking forward, we plan to apply our adaptive tensor memoization algorithm to other tensor decompositions, such as CP-APR. We also believe a closer inspection of not just the arithmetic but also communication properties of our method, coupled with more architecture-specific tuning, manycore co-processor acceleration, and extensions for distributed memory, are ripe opportunities for future work.

Acknowledgment

We thank Piyush Sao, Casey Battaglino, and Xiaolong Wu for their comments. This material is based upon work supported by the U.S. National Science Foundation (NSF) Award Number 1533768.

References

[1] A. Novikov, D. Podoprikhin, A. Osokin, and D. Vetrov, "Tensorizing neural networks," *CoRR*, vol. abs/1509.06569, 2015.

[2] I. Perros, R. Chen, R. Vuduc, and J. Sun, "Sparse hierarchical tucker factorization and its application to healthcare," *IEEE International Conference on Data Mining (ICDM)*, 2015.

[3] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, 2009.

[4] A. Cichocki, "Era of big data processing: A new approach via tensor networks and tensor decompositions," *CoRR*, vol. abs/1403.2048, 2014.

[5] L. De Lathauwer and D. Nion, "Decompositions of a higher-order tensor in block terms—Part III: Alternating least squares algorithms," *SIAM Journal on Matrix Analysis and Applications*, vol. 30, no. 3, pp. 1067–1083, 2008.

[6] J. Li, Y. Ma, C. Yan, and R. Vuduc, "Optimizing sparse tensor times matrix on multi-core and many-core architectures," in *ACM/IEEE the Sixth Workshop on Irregular Applications: Architectures and Algorithms*. Salt Lake City, Utah, USA: IEEE, 2016.

[7] J. Li, C. Battaglino, I. Perros, J. Sun, and R. Vuduc, "An input-adaptive and in-place approach to dense tensor-times-matrix multiply," in *ACM/IEEE Supercomputing (SC '15)*. New York, NY, USA: ACM, 2015.

[8] S. Smith, N. Ravindran, N. Sidiropoulos, and G. Karypis, "Splatt: Efficient and parallel sparse tensor-matrix multiplication," in *Proceedings of the 29th IEEE International Parallel & Distributed Processing Symposium*, ser. IPDPS, 2015.

[9] J. C. Ho, J. Ghosh, and J. Sun, "Marble: High-throughput phenotyping from electronic health records via sparse nonnegative tensor factorization," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '14. New York, NY, USA: ACM, 2014, pp. 115–124.

[10] Y. Wang, R. Chen, J. Ghosh, J. C. Denny, A. Kho, Y. Chen, B. A. Malin, and J. Sun, "Rubik: Knowledge guided tensor factorization and completion for health data analytics," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '15. New York, NY, USA: ACM, 2015, pp. 1265–1274.

[11] U. Kang, E. Papalexakis, A. Harpale, and C. Faloutsos, "GigaTensor: Scaling tensor analysis up by 100 times—algorithms and discoveries," in *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '12. New York, NY, USA: ACM, 2012, pp. 316–324.

[12] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org.

[13] A. Anandkumar, R. Ge, D. Hsu, S. M. Kakade, and M. Telgarsky, "Tensor decompositions for learning latent variable models," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 2773–2832, Jan. 2014.

[14] E. E. Papalexakis, C. Faloutsos, and N. D. Sidiropoulos, "ParCube: Sparse parallelizable tensor decompositions," in *Proceedings of the 2012 European Conference on Machine Learning and Knowledge Discovery in Databases - Volume Part I*, ser. ECML PKDD'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 521–536.

[15] L. Grasedyck, D. Kressner, and C. Tobler, "A literature survey of low-rank tensor approximation techniques," *GAMM-Mitteilungen*, vol. 36, no. 1, pp. 53–78, 2013.

[16] A. Cichocki, N. Lee, I. V. Oseledets, A. Phan, Q. Zhao, and D. Mandic, "Low-rank tensor networks for dimensionality reduction and large-scale optimization problems: Perspectives and challenges part 1," *ArXiv e-prints*, Sep. 2016.

[17] A. H. Phan, P. Tichavsk, and A. Cichocki, "Fast alternating LS algorithms for high order CANDECOMP/PARAFAC tensor factorizations," *IEEE Transactions on Signal Processing*, vol. 61, no. 19, pp. 4834–4846, Oct 2013.

[18] F. Huang, N. Niranjan U., I. Perros, R. Chen, J. Sun, and A. Anandkumar, "Scalable Latent Tree Model and its Application to Health Analytics," *ArXiv e-prints*, Jun. 2014.

[19] O. Kaya and B. Uçar, "Scalable sparse tensor decompositions in distributed memory systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '15. New York, NY, USA: ACM, 2015, pp. 77:1–77:11.

[20] J. H. Choi and S. Vishwanathan, "Dfacto: Distributed factorization of tensors," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 1296–1304.

[21] B. W. Bader and T. G. Kolda, "Efficient MATLAB computations with sparse and factored tensors," *SIAM Journal on Scientific Computing*, vol. 30, no. 1, pp. 205–231, December 2007.

[22] B. W. Bader, T. G. Kolda *et al.*, "MATLAB Tensor Toolbox (Version 2.6)," Available online, February 2015.

[23] S. Smith and G. Karypis, "Tensor-matrix products with a compressed sparse tensor," in *Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms*. ACM, 2015, p. 7.

[24] J. Carroll and J.-J. Chang, "Analysis of individual differences in multidimensional scaling via an n-way generalization of eckart-young decomposition," *Psychometrika*, vol. 35, no. 3, pp. 283–319, 1970.

[25] S. Zhou, X. V. Nguyen, J. Bailey, Y. Jia, and I. Davidson, "Accelerating online CP decompositions for higher order tensors," *22th ACM SIGKDD 2016 Conference on Knowledge Discovery & Data Mining (submitted)*, Aug 2016.

[26] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. Hruschka, and T. Mitchell, "Toward an architecture for never-ending language learning," 2010.

[27] O. Görlitz, S. Sizov, and S. Staab, "PINTS: Peer-to-peer infrastructure for tagging systems," in *Proceedings of the 7th International Conference on Peer-to-peer Systems*, ser. IPTPS'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 19–19.

[28] S. Smith, J. W. Choi, J. Li, R. Vuduc, J. Park, X. Liu, and G. Karypis. (2017) FROSTT: The formidable repository of open sparse tensors and tools. [Online]. Available: http://frostt.io/

[29] S. Smith, N. Ravindran, N. Sidiropoulos, and G. Karypis, "SPLATT: The Surprisingly ParalleL spArse Tensor Toolkit (Version 1.1.1)," Available online, 2016.

[30] E. C. Chi and T. G. Kolda, "On tensors, sparsity, and nonnegative factorizations," *SIAM Journal on Matrix Analysis and Applications*, vol. 33, no. 4, pp. 1272–1299, 2012.

[31] L. Sorber, M. Van Barel, and L. De Lathauwer, "Tensorlab (Version v3.0)," Available online, 2014.

[32] M. Baskaran, B. Meister, N. Vasilache, and R. Lethin, "Efficient and scalable computations with sparse tensors," in *High Performance Extreme Computing (HPEC), 2012 IEEE Conference on*, Sept 2012, pp. 1–6.

[33] O. Kaya and B. Uar, "High-performance parallel algorithms for the Tucker decomposition of higher order sparse tensors." *Inria - Research Centre Grenoble Rhone-Alpes*, RR-8801, 2015.

[34] ——, "Parallel cp decomposition of sparse tensors using dimension trees." *Inria - Research Centre Grenoble Rhone-Alpes*, RR-8976, 2016.